Alma Mater Studiorum · Università di Bologna

Corso di Laurea in Ingegneria e Scienze informatiche

CHATBOT PER LA MEMORIZZAZIONE E LA RICERCA DI INFORMAZIONI MULTIMEDIALI CON INDICIZZAZIONE SEMI-AUTOMATICA

Tesi di Laurea in Algoritmi e strutture dati

Relatore:
Prof. Vittorio Maniezzo

Presentata da:
Alex Ravaglia

Correlatore:
Dott. Michele Amati

II Sessione
Anno Accademico 2017/2018

Sommario

Lo scopo principale della seguente tesi è quello di progettare e sviluppare un chatbot che archivi fotografie in modo intelligente. Le immagini che dovrà archiviare sono di un tipo specifico, principalmente saranno foto di post-it e simili. L'idea è quella di creare un chatbot che memorizzi e gestisca questo tipo di dato. Il sistema estrae una serie di informazioni in modo automatico, risparmiando all'utente di dovere compiere tale operazione. In futuro l'utente dovrà essere in grado di recuperare le proprie immagini rispondendo a una serie di domande. La chat è fondamentale poiché è uno strumento che permette all'utente di esprimersi in maniera naturale e con immediatezza. L'usabilità e la User Experience del sistema devono permettere che queste potenzialità siano sfruttate al massimo.

Indice

Capitolo 1

Introduzione

L'obiettivo di questa tesi è quello di risolvere un problema pratico mediante la progettazione e implementazione di un Chatbot. Nella fase preliminare alla progettazione è stato individuato il problema per il quale si è deciso di implementare un servizio automatizzato volto a risolverlo. Tale problema viene presentato nel seguente scenario: capita spesso di volere memorizzare sul proprio smartphone una serie di informazioni scritte su bigliettini, agende o post-it in modo da averli sempre con sé. Questi dati potrebbero essere un indirizzo civico, un numero di telefono, la password del wi-fi o di qualsiasi altro genere. Ciò che li accomuna è il fatto che spesso questi dati vengono appuntati o scritti su foglietti di carta e può capitare di perderli o non ricordare dove si sono lasciati. L'esigenza di volere avere questo tipo di dato a portata di mano, porta le persone a scattare una foto e custodirla all'interno del proprio smartphone. La 'galleria fotografica' raccoglie le fotografie, si pensi che sia destinata a memorizzare immagini di paesaggi, foto di amici, della famiglia, del proprio cane e non l'immagine di un post-it che ricorda un indirizzo o la password del wi-fi. La natura delle motivazioni che portano a scattare queste tipologie di foto non è la stessa, quindi può essere comodo volerle memorizzare in ambiti separati. La memorizzazione di tutte le foto di post-it e bigliettini in una stessa directory separata potrebbe essere una

buona soluzione. Non ci vorrà molto che questo raccoglitore si riempia di immagini e la ricerca risulti particolarmente scomoda. Da qui viene l'esigenza di produrre un sistema tramite l'ausilio di un chatbot che riesca ad aiutare l'utente ad archiviare tutte le proprie fotografie. Tra le operazioni più interessanti vi sono quelle di archiviazione e recupero di un'immagine. Queste funzionalità devono permettere all'utente di svolgerle con estrema praticità ed immediatezza; in quest'ottica si è pensato allo sviluppo di un sistema che effettui un tagging implicito per ogni immagine ricevuta. L'idea è quella di permettere all'utente di inviare un'immagine e riuscire a recuperarla nel momento del bisogno. Dalle foto che il sistema riceve, vengono estratte una serie di informazioni senza che sia l'utente a fornirle. Si possono estrarre dati come la posizione, la data, l'ora, il colore del post-it. Per effettuare una ricerca tra le proprie foto, sarà necessario rispondere a una serie di domande che il sistema pone. In questo modo il bot riesce a ricercare tra tutte le immagini quella richiesta dall'utente. Durante la fase di ricerca, il bot possiede una serie di informazioni per ogni foto, per questo motivo basa le domande da porre all'utente in funzione dei dati estratti. Si è cercato in letteratura se esistessero applicazioni volte a risolvere tale problema, ma nessuna in particolare offriva una soluzione specifica al problema individuato. La decisione di utilizzare la chat è stata fondamentale. Infatti è lo strumento ideale per lo sviluppo di risoluzione di un problema di questo tipo perché offre un ambiente nel quale l'utente può esprimersi con facilità e naturalezza. Un chatbot è un agente di conversazione che interagisce con gli utenti in un certo dominio o su un determinato argomento utilizzando il linguaggio naturale. Normalmente un chatbot risponde a una richiesta dell'utente che può essere una domanda o un commento. Tra le potenzialità che si possono sfruttare in questo ambiente vi può essere la possibilità che il bot riesca a interpretare il linguaggio naturale (NLP). Una persona potrebbe esprimersi in modo naturale comunicando al chatbot le proprie intenzioni. Non appena il sistema ha compreso la natura delle richieste

dell'utente, cerchi di risolverle e comunicarle mediante l'uso di una chat. L'utente non deve avere bisogno di particolari competenze tecnologiche, è richiesto solamente che sia in grado di utilizzare un'applicazione di messaggistica. La realizzazione del progetto si è divisa in: analisi del problema, progettazione di una soluzione e realizzazione vera e propria. Durante la fase di analisi si è studiato il problema, sono state fatte delle valutazioni sul dominio applicativo ed è stata formulata la specifica dei requisiti. Nella fase di progettazione sono state scelte le tecnologie considerate più idonee, identificata l'architettura del sistema e della base dati. È seguita la fase di implementazione vera e propria con scrittura del codice e test finali.

Capitolo 2

Scenario

2.1 Un problema da risolvere

Spesso può capitare di volere memorizzare sul proprio smartphone una serie di informazioni scritte su bigliettini, agende o post-it in modo da averle sempre con sé. I dati che si vogliono memorizzare potrebbero essere un indirizzo civico, un numero di telefono, uno scontrino, la password del Wi-fi di casa o dell'ufficio. Può capitare che questi dati siano appuntati su foglietti di carta 'volanti' e spesso si finisce per perderli o non ricordare dove si sono lasciati. L'esigenza di volere avere questo tipo di dato a portata di mano, porta le persone a scattare una foto e custodirla all'interno dello smartphone. La 'galleria' del proprio device è destinata a raccogliere le foto che l'utente scatta durante la giornata, idealmente è destinata a memorizzare immagini di paesaggi, foto di amici, della famiglia, del proprio cane e non l'immagine di un post-it che ricorda un indirizzo o la password del Wi-fi. La natura delle motivazioni che portano a scattare queste tipologie di foto non è la stessa, quindi può essere comodo volerle memorizzare in ambiti separati. Si potrebbe decidere di memorizzare tutte le foto di post-it in una stessa directory separata dalle altre immagini, ma non ci vorrà molto tempo che essa si riempi di foto e la ricerca diventi un

compito oneroso. Idealemente, la fase di memorizzazione dovrebbe consistere solamente nello scattare una foto e aggiungere qualche tag. Per la fase di ricerca si dovrebbe arrivare al risultato specificando solamente il tipo di informazione che si vuole recuperare. Sarebbe interessante riuscire ad avere un sistema dove l'operazione di salvataggio viene semplificata al punto che si richieda solamente di scattare una foto senza per forza dover specificare la tipologia del contenuto informativo. Il problema si può quindi riassumere come l'esigenza di:

- memorizzazione di una serie di informazioni e averle sempre a portata di mano
- recupero di tali informazioni al momento del bisogno
- la memorizzazione deve essere veloce e pratica
- la fase di ricerca deve fornire il risultato corretto senza richiedere all'utente troppe informazioni
- avere un sistema che gestisca al meglio e riorganizzi i dati

2.2 Soluzioni

2.2.1 Soluzioni esistenti

Essendo un problema pratico si è cercato di capire come solitamente questa esigenza venisse risolta quotidianamente. Sono state individuate varie soluzioni:

- Utilizzo di applicazioni apposite(Evernote, Microsoft Onenote, Google Keep ecc.)
- Utilizzo di una chat 'ribattezzata' come contenitore di foto
- Memorizzazione nella galleria dello smartphone

– Memorizzazione su cloud

2.2.2 Problemi

Nessuna delle soluzioni individuate risolve tutti i problemi preposti. La memorizzazione su cloud così come l'invio delle proprie foto in una chat e il salvataggio nel dispositivo, permettono di avere un modo veloce per l'archiviazione, ma non forniscono altrettanta efficienza in fase di recupero. Sarà necessario scorrere tutte le immagini salvate e recuperare quella d'interesse. Le applicazioni già esistenti invece forniscono un approccio valido alla risoluzione dei problemi di memorizzazione e recupero, ci si è allora domandati se non esistesse un metodo più pratico, che non richiedesse di dovere scaricare, installare e imparare a usare una nuova applicazione.

2.2.3 Soluzione individuata

Si è individuato uno strumento di cui l'utente conosce già il funzionamento e utilizza abitualmente, la chat. Avendo trovato un'infrastruttura ideale si è deciso di implementare il servizio tramite un chatbot.

2.3 Chatbot

Un chatbot è un agente di conversazione che interagisce con gli utenti in un certo dominio o su un determinato argomento utilizzando il linguaggio naturale. Normalmente un chatbot risponde a una richiesta dell'utente che può essere una domanda o un commento. Tra gli esempi di chatbot più famosi ci sono Siri e Cortana, assistenti personali messi a disposizione rispettivamente da Apple e Microsoft. La nascita dei chatbot risale al matematico Alan Turing il quale nel 1950 pubblicò un articolo dal titolo "Computing

Machinery and Intelligence", in cui propose un criterio, oggi definito Test di Turing, in grado di determinare se una macchina fosse in grado di pensare o meno. Per soddisfare questo criterio, un software deve fingere di essere umano in una conversazione in tempo reale in modo che l'interlocutore non sia in grado di distinguere, basandosi solo sul contenuto della conversazione, se stia conversando con un programma o con un essere umano. In un passato più recente, era un bot Clippy, la graffetta di Microsoft Office che interagiva con l'utente, forniva suggerimenti anche quando non richiesti e rispondeva alle domande. Fino ad allora, però, i bot non avevano avuto particolare fortuna per via della mancanza di un'infrastruttura che fosse largamente utilizzata e che supportasse questa tecnologia. Con il passare del tempo l'infrastruttura che mancava è stata identificata nelle app di messaggistica che hanno preso sempre più piede negli ultimi anni, basti pensare che più di 2,5 miliardi di persone utilizzano almeno un'app per comunicare [1]. La prima app di messaggistica a introdurre i bot è stata Telegram, successivamente anche Facebook Messenger ha integrato la possibilità di dialogare con chatbot. Gli impieghi possibili per i chatbot sono molteplici: dal customer care, alla diffusione di notizie, offerte e promozioni, supporto nell'acquisto su un e-Commerce, attivazione di un servizio, assistenza personale e tanti altri. L'ultima importante evoluzione è stata quella di integrare i pagamenti. In questo modo le aziende hanno la possibilità di vendere prodotti e servizi ai loro clienti direttamente dalla chat, senza che gli utenti lascino l'app di messaggistica per fare i loro acquisti.

[1] `https://www.softfobia.com/news/mobile/13/cosa-sono-i-chatbot-come\-funzionano-intelligenza-artificiale-marketing`

Figura 2.1: Immagine - Chatbot

2.4 Perchè un chatbot?

2.4.1 Vantaggi per l'utente

Il chatbot risiede all'interno di un'applicazione di messaggistica quale Telegram o Facebook Messenger. Il vantaggio è che queste applicazioni sono conosciute e utilizzate abitualmente da milioni di utenti attivi al mese. In questo modo il servizio che si cerca di offrire non deve essere installato da fonti esterne ma lo si può trovare integrato nelle applicazioni che un utente è solito usare. In questo modo non vi è il bisogno di dovere imparare a utilizzare un nuovo strumento tecnologico. Rimanendo in un contesto noto, la chat, sarà più immediato interfacciarsi con questo nuovo servizio offerto.

2.4.2 Vantaggi tecnologici

2.4.3 Elaborazione del linguaggio naturale

L'elaborazione del linguaggio naturale, detta anche NLP (dall'inglese Natural Language Processing, elaborazione lingua naturale), è il processo di trattamento automatico mediante un calcolatore elettronico, delle informazioni scritte o parlate in una lingua naturale. Questo processo è reso particolarmente difficile e complesso a causa delle caratteristiche intrinseche di ambiguità del linguaggio umano. Per questo motivo il processo di elaborazione viene suddiviso in fasi diverse, tuttavia simili a quelle che si possono incontrare nel processo di elaborazione di un linguaggio di programmazione:

- Analisi lessicale: scomposizione di un'espressione linguistica in token (in questo caso le parole)
- Analisi grammaticale: associazione delle parti del discorso a ciascuna parola nel testo
- Analisi sintattica: arrangiamento dei token in una struttura sintattica (ad albero: parse tree)
- Analisi semantica: assegnazione di un significato (semantica) alla struttura sintattica e, di conseguenza, all'espressione linguistica.

2.4.4 Vantaggi dell'elaborazione del linguaggio naturale

Grazie all'utilizzo della chat l'utente può esprimersi come farebbe in una conversazione con un altro essere umano. Il bot implementa forme di Intelligenza artificiale e interpretazione del linguaggio naturale in modo che il servizio che offre diventi notevolmente più interessante per l'utente, il quale può avere a disposizione un servizio personalizzato, efficiente e immediato con il quale può conversare come se lo stesse facendo con un umano. È

inoltre possibile prevedere ciò che l'utente richiede e fornirgli una risposta personalizzata anche per le richieste più imprevedibili.

2.4.5 Svantaggi dell'elaborazione del linguaggio naturale

L'interpretazione del linguaggio naturale è un problema difficile. Quando si cerca di illudere l'utente che lo scambio di messaggi sia con un umano e non con una macchina, si potrebbe verificare il fatto che la conversazione prenda una direzione troppo colloquiale e spontanea. In questo modo l'interpretazione dei messaggi ricevuti diventa più complicata e inevitabilmente si alza il margine di errore.

2.4.6 Vincoli

La decisione di sviluppare il servizio come chatbot lega inevitabilmente il bot all'applicazione di messaggistica designata. Se il servizio viene implementato su telegram, Il quale in modo autonomo decide di non volere supportare più i bot, l'intero sistema non sarà più disponibile. Sarà necessario migrare su un'altra piattaforma (es. Facebook Messenger) riadattando il bot alle policy e vincoli che ogni piattaforma impone.

2.5 Applicazione di messaggistica

Tra le applicazioni di messaggistica più utilizzate al giorno d'oggi che danno la possibilità di integrare con i chatbot ci sono Facebook Messenger e Telegram. La decisione su quale piattaforma scegliere è ricaduta su Telegram poiché ha integrato l'uso dei bot prima di Messenger e sul web si è creata una community di sviluppatori molto più attiva rispetto quella degli sviluppatori per Facebook. Telegram perdipiù mette a disposizione le proprie API e protocolli per chiunque voglia implementare un client Telegram in autonomia. Con-

siderando la vasta offerta di API e librerie per la creazione di chatbot messe a disposizione per gli sviluppatori, si è identificato in Telegram lo strumento ideale sul quale appoggiarsi per l'implementazione del sistema.

2.6 Chat

Una chat costituisce un sistema di comunicazione in modalità sincrona che permette a più utenti collegati nello stesso momento di scambiarsi brevi messaggi scritti, emulando una conversazione. Tramite una connessione ad Internet la comunicazione avviene in tempo reale. Per entrare a far parte di una chat, occorre che si utilizzi un'applicazione adatta come Facebook Messenger, WhatsApp o Telegram. Per iniziare a chattare con qualcuno, è richiesta una registrazione e la conoscenza del contatto dell'utente con il quale ci si vuole connettere.
L'idea è quella di sfruttare questo potente strumento di comunicazione utilizzato da molte persone e fare in modo che l'utente al posto di conversare con un altro essere umano, inizi la conversazione con un sistema automatico.

2.7 Telegram

Telegram è un servizio di messaggistica istantanea basato su cloud ed erogato senza fini di lucro dalla società Telegram LLC. I client ufficiali di Telegram sono distribuiti come software libero per Android, GNU/Linux, iOS, MacOS, Windows NT e Windows Phone. Caratteristiche di Telegram sono la possibilità di stabilire conversazioni tra due o più utenti, effettuare chiamate vocali cifrate "punto-punto", scambiare messaggi vocali, videomessaggi, fotografie, video, stickers e file di qualsiasi tipo grandi fino a 1,5 GB. Telegram è stato fondato nel 2013 dai fratelli Nikolai e Pavel Durov. Il protocollo sul quale si basa Telegram

è MTProto ed è stato scritto da Nikolai Durov. È considerata un'applicazione robusta e sicura, nel 2018 conta 200 milioni di utenti attivi al mese.

2.8 Telegram-Bot

Da giugno 2015 Telegram ha introdotto una piattaforma per permettere, a sviluppatori terzi, di creare i Bot. I Bot sono degli account Telegram, gestiti da un programma, offrono molteplici funzionalità con risposte immediate e completamente automatizzate, non richiedono un numero di telefono aggiuntivo da configurare. Gli utenti possono interagire con i bot in 2 modi, aprendo una chat con loro, o aggiungendoli ai gruppi. I messaggi e comandi inviati dall'utente vengono inviati e processati dal software in esecuzione sul server del bot, la comunicazione avviene tramite un'interfaccia HTTPS. Dal 4 gennaio 2016 è disponibile una nuova modalità: Inlinebot, permette di utilizzare un bot semplicemente citandolo con il proprio username in qualsiasi chat (chat cloud, canali e gruppi). Per creare e gestire i propri bot è stato creato Botfather (@BotFather). Telegram mette a disposizione una serie di API per gli sviluppatori che vogliono implementare chatbot, sono disponibili anche librerie e framework di terze parti. L'utente all'interno della chat con il bot può comunicare in diversi modi, esprimendosi con il linguaggio naturale, interagendo su bottoni che il bot mette in evidenza sullo schermo o nella tastiera, impartendo una serie di domandi definiti a priori che il bot riconosce. Per impartire un comando si digiti: '/nomeComando'.

Figura 2.2: Immagine - Telegram Chatbot

2.9 Python

Python è un linguaggio di programmazione dinamico ad alto livello, orientato agli oggetti. Offre un forte supporto all'integrazione con altri linguaggi e programmi, è fornito di una estesa libreria standard. È stato rilasciato pubblicamente per la prima volta nel 1991 dal suo creatore Guido van Rossum, supporta diversi paradigmi di programmazione, come quello object-oriented (con supporto all'ereditarietà multipla), quello imperativo e quello funzionale. Offre una tipizzazione dinamica forte, il controllo dei tipi viene eseguito a runtime (dynamic typing). Una variabile è un contenitore al quale viene associata un'etichetta (il nome) che può essere associata a diversi contenitori anche di tipo diverso durante il suo tempo di vita. Usa un garbage collector per la liberazione automatica della memoria. È fornito di una libreria built-in estremamente ricca, la sintassi è pulita così come i suoi costrutti, i blocchi logici vengono costruiti semplicemente allineando le righe allo stesso modo, incrementando la leggibilità e l'uniformità del codice. Python è un linguaggio pseudocompilato: un interprete si occupa di analizzare il codice sorgente (semplici file testuali con estensione .py) e, se sintatticamente corretto, di eseguirlo. In Python, non esiste una fase di compilazione separata che generi un file eseguibile partendo dal sorgente. L'esser pseudointerpretato rende Python un linguaggio portabile. Una volta scritto un sorgente, esso può essere interpretato ed eseguito sulla gran parte delle piattaforme attualmente utilizzate, siano esse di casa Apple (Mac) che PC (Microsoft Windows e GNU/Linux), semplicemente, basta la presenza della versione corretta dell'interprete. Si è deciso di utilizzare questo linguaggio per le potenzialità appena citate, in particolare perché è considerato tra i migliori linguaggi per l'implementazione di bot telegram, sul web vanta di community numerose e attive in questo campo.

2.9.1 Python-telegram-bot

È una libreria che fornisce un'interfaccia Python per l'utilizzo delle API messe a disposizione da Telegram per l'implementazione di Bot. È compatibile con le versioni Python 2.7, 3.3+ e PyPy. Oltre alla pura implementazione delle API, la libreria presenta una serie di classi di alto livello per rendere lo sviluppo di bot semplice e immediato. Queste classi sono contenute nel sottomodulo 'telegram.ext'. La libreria è rilasciata sotto licenza LGPL-3.

2.9.2 PyTransitions

Pytransitions è una libreria che permette di implementare una macchina a stati finiti in un contesto orientato agli oggetti in Python. L'attenzione è focalizzata sugli stati e sulle transizioni. Vi è la possibilità di definire un comportamento in entrata/uscita da ogni stato, valutare le transizioni in funzione degli eventi ricevuti e definire lo stato di arrivo.

2.10 Eclipse & PyDev

Eclipse è un ambiente di sviluppo integrato ed è l'IDE Java più utilizzato. Contiene uno spazio di lavoro di base e un sistema plug-in estendibile per la personalizzazione dell'ambiente. Può essere usato per sviluppare applicazioni in svariati linguaggi di programmazione tramite l'installazione di plug-in specifici come PyDev.

pydev: PyDev è un plug-in di terze parti per Eclipse. È un ambiente di sviluppo integrato (IDE) per il linguaggio Python. Supporta, tra le altre opzioni, il refactoring del codice, la programmazione di interfacce grafiche, il debugging e l'analisi del codice ed è open source.

2.11 Reverse Geocoding

Il Geocoding è una tecnica che consente di convertire l'indirizzo di una via, una piazza o più semplicemente del centro città, nelle corrispondenti coordinate geografiche di latitudine e longitudine. Il problema inverso viene definito come Reverse geocoding e consente di risalire all'indirizzo o città di un luogo partendo dalle sue coordinate GPS (latutidine-longitudine).

2.12 Mapbox

È un servizio online che fornisce un supporto tramite la chiamata di specifiche API a problemi che utilizzano coordinate GPS per calcolare indirizzi, distanze ed elaborazione di mappe. Il servizio è gratuito per un numero limitato di chiamate mensili, successivamente diventa a pagamento in funzione di quanto viene utilizzato. Per potere invocare le API è necessario inviare congiuntamente alla richiesta, un token univoco che viene rilasciato allo sviluppatore in fase di registrazione. Vengono utilizzati i servizi di Mapbox per le API particolarmente efficienti sul problema di 'reverse geocoding'.

Capitolo 3

Analisi

3.1 Dominio Applicativo

Gli scenari in cui il sistema verrà utilizzato appartengono alla quotidianità di ogni persona. Quando quest'ulitma avrà la necessità di volere archiviare una propria immagine, gli basterà scattare una foto e inviarla nella chat. Allo stesso modo quando avrà bisogno di recuperarla, consulti la chat e si faccia inviare l'immagine voluta. La tipologia delle foto che vengono inviate al sistema è molto specifica, infatti vi dovrebbero essere esclusivamente immagni di foglietti di carta , cartellini, etichette, scontrini e post-it. L'utenza alle quali il servizio è rivolto è tra le più svariate, non vi è un target specifico, questo perché il problema che si propone di risolvere è abbastanza comune. L'unica considerazione che si può fare sul dominio applicativo è il fatto che il sistema potrebbe essere utilizzato in situazioni in cui l'utente non abbia molto tempo a disposizione o non voglia dedicargliene troppo. Essendo implementato all'interno di una chat è requisito fondamentale che gli utenti sappiano interfacciarsi con un'applicazione di messaggistica.

3.2 Analisi dei requisiti

L'analisi dei requisiti è un'attività preliminare allo sviluppo del chatbot, il cui scopo è quello di definire le funzionalità e le proprietà che il bot deve offrire. Guida le fasi successive di sviluppo, che complessivamente sono volte a realizzare quanto previsto da tale specifica. Il documento principale che viene prodotto dall'analisi dei requisiti è la specifica dei requisiti dove vengono definiti i requisiti funzionali e non funzionali.

3.3 Specifica dei requisiti

3.3.1 Requisiti funzionali

I requisiti funzionali sono un elenco delle funzionalità che il sistema deve avere. Il chatbot dovrà consentire all'utente di svolgere le seguenti operazioni:

- Inserimento di una foto: dovrà essere possibile inserire e inviare una foto al bot.
- Memorizzazione delle foto: La foto ricevuta deve essere corredata da una serie di informazioni che si deve dare modo all'utente di inserire, quali: posizione, parole chiave o nessun tipo di dato.
- Recupero della foto: La foto che un utente vuole recuperare può essere specificata attraverso una serie variabile di parametri quali posizione, parole chiave o nessun paramento in particolare.
- Eliminazione di una foto aggiunta: L'utente deve riuscire a eliminare una foto che ha precedentemente inserito.
- Registrazione di un nuovo utente.

3.3.2 Requisiti non funzionali

I requisiti non funzionali rappresentano i vincoli e le proprietà relative al sistema. Non sono un elenco di funzionalità che devono essere implementate ma possono essere vincoli di natura temporale o vincoli sulla qualità. I requisiti non funzionali più critici che il chatbot dovrà rispettare sono:

Usabilità

- Velocità: il sistema deve essere veloce da utilizzare.
- Apprendibilità: deve essere semplice imparare ad utilizzare il sistema.
- Facilità d'uso: la navigazione e l'utilizzo del bot non devono richiedere conoscenze che vanno oltre il sapere utilizzare una chat.

Performance

- Tempo di risposta: Il tempo di risposta che il sistema impiega per rispondere a un utente deve essere breve, la risposta deve essere quasi immediata.

Requisiti non funzionali, meno critici:

Portabilità Il sistema è dipendente da un'applicazione di messagistica, nell'ottica di possibili migrazioni future o necessità di espansione su altre piattaforme è necessario ridurre per quanto possibile le dipendenze con questa applicazione.

Affidabilità Il sistema dovrà essere affidabile gestendo le situazioni di errore.

3.3.3 Requisiti essenziali

Per creare un prodotto diverso dalle applicazioni già esistenti, le quali risolvono un problema simile a quello che ci siamo preposti, si è deciso di evidenziare i punti di forza del sistema:

- L'utente deve riuscire a recuperare le proprie foto anche se nella fase di invio dell'immagine non ha inserito alcuna parola chiave o informazione correlata a quella foto.

- L'implementazione del bot all'interno di una chat sfrutti le conoscenze che un utente già possiede nell'usarla. In quest'ottica le operazioni di inserimento devono essere estremamente immediate evitando una navigazione profonda e complessa. Nelle operazioni di ricerca l'utente può essere indotto a digitare qualche messaggio in più rispetto la fase di inserimento, l'esperienza di navigazione dell'utente non deve essere troppo complessa e articolata.

3.3.4 Requisiti tecnologici

Il sistema essendo implementato come chatbot all'interno di Telegram prevede che l'applicazione sia già installata e l'utente la sappia utilizzare.

3.4 Diagramma dei casi d'uso

Lo studio dei requisiti appena presentati ha permesso la creazione del seguente diagramma dei casi d'uso, dove si possono vedere le principali funzionalità che l'utente deve riuscire a portare a termine.

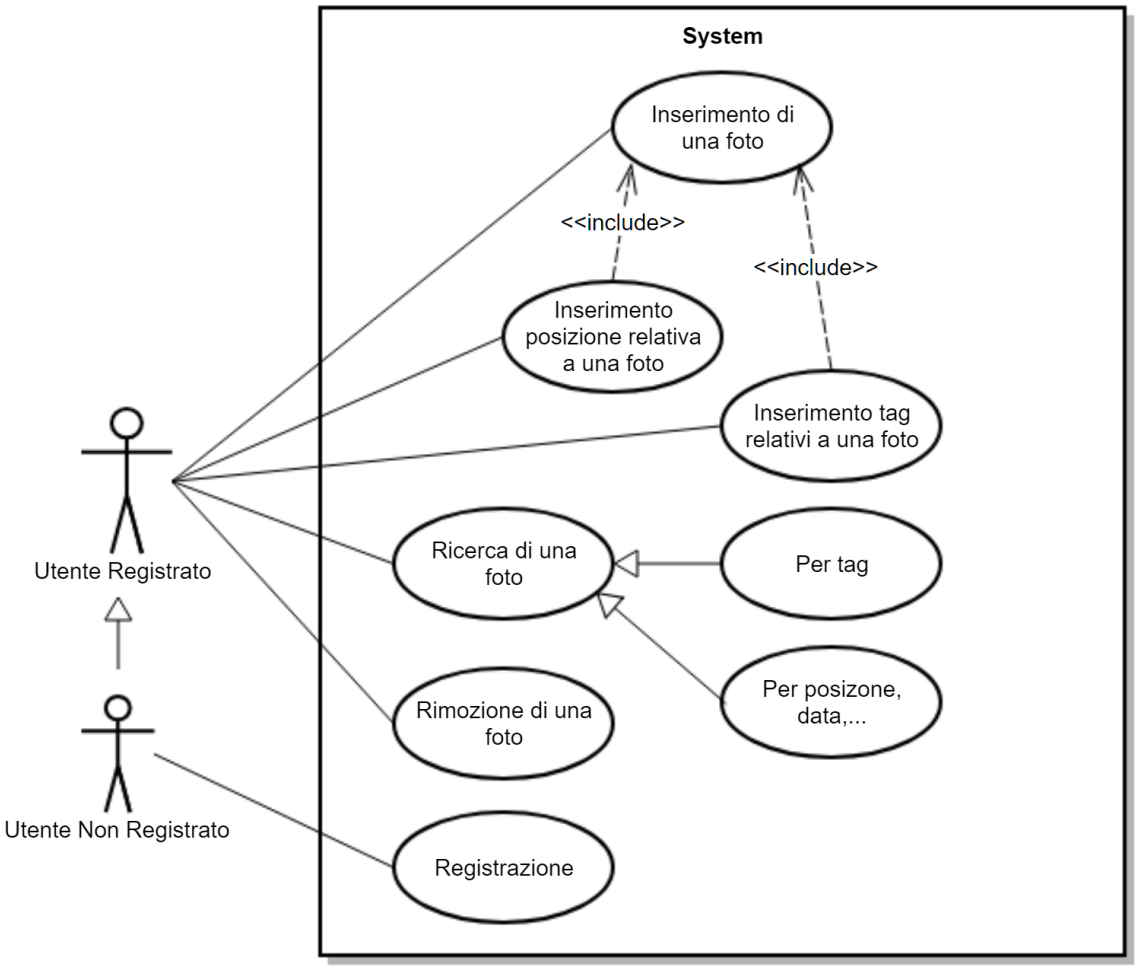

Figura 3.1: Sistema - Diagramma dei casi d'uso

Capitolo 4

Progettazione del sistema

4.1 Architettura del sistema

Per la realizzazione del progetto sulla base delle specifiche formulate, si sono individuate varie entità fisiche sulle quali avviene l'esecuzione di diverse parti del sistema. Vi sono 4 macro entità principali:

- Smartphone dell'utente: Dispositivo nel quale è installata l'applicazione di messaggistica (Telegram). L'utente tramite questa applicazione può leggere i messaggi che gli arrivano e inviare comandi al bot.
- Server di Telegram: Intermediario tra i messaggi inviati dall'utente e il server dove risiede il bot, tutto il traffico dati passa attraverso questo nodo.
- Server del bot: È il cuore dell'applicazione, elabora le richieste ricevute dall'utente, vengono eseguiti gli algoritmi di memorizzazione, estrazione delle informazioni, interpretazione dei messaggi e tutte le funzionalità che il bot possiede.

– Database: Archivio dove vengono memorizzate tutte le informazioni relative a ogni utente (foto, tag, informazioni varie...).

Le relazioni dei vari nodi si possono vedere nel seguente diagramma di deployment.

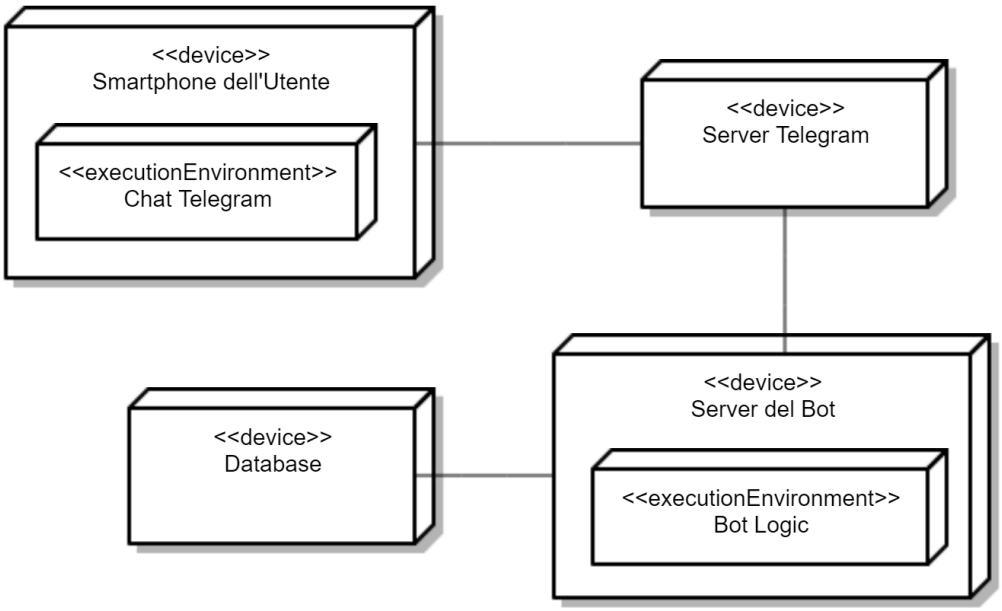

Figura 4.1: Sistema - Diagramma di deployment

4.2 Progettazione dei vari componenti

La progettazione del sistema ha richiesto lo sviluppo di 2 delle 4 entità appena descritte, nello specifico il Server del Bot e il Database. Non si ha nessun tipo di controllo sull'applicazione di messaggistica in esecuzione sullo smartphone dell'utente e nemmeno sul server di Telegram. Per tale motivo si deve riuscire a realizzare un sistema che si adatti alle regole di comunicazioni messe a disposizione da Telegram.

4.3 Comunicazione Telegram/Bot e aggiornamenti

Ci sono due modi per ricevere gli aggiornamenti che arrivano al bot che si escludono a vicenda: il metodo Long Polling e il metodo Webhooks. Gli aggiornamenti in arrivo vengono memorizzati sul server di Telegram fino a quando il bot non li riceve (in un modo o nell'altro) ma non vengono memorizzati per più di 24 ore. Indipendentemente dall'opzione scelta si ricevono oggetti di aggiornamento serializzati JSON. La decisione, che è stata presa in fase di progettazione, è stata quella di realizzare un servizio LongPolling. Nei prossimi paragrafi vengono spiegati i due metodi di comunicazione; ci si riferisce con il termine server al server di Telegram, mentre con Client al bot.

4.3.1 Whebooks

Ogni volta che c'è un aggiornamento per il bot sui Server di Telegram, viene inviata una richiesta POST HTTPS all' URL specificato sul quale è attivo il servizio del bot, contenente un aggiornamento serializzato JSON. In caso di una richiesta non riuscita, si tenta più volte prima di rinunciare a comunicare l'aggiornamento. Se si desidera assicurarsi che la richiesta di Webhook provenga da Telegram, si utilizzi un percorso segreto nell'URL, ad es. https://www.example.com/ <token>. Considerando che nessun altro conosce il token del bot, si è sicuri che la richiesta provenga da Telegram. Per implementare gli aggiornamenti tramite weebboks è necessario avere un server[1] che:

- Supporti IPv4
- Accetti in entrata messaggi POST da 149.154.167.197-233 sulle porte 443,80,88 o 8443
- Gestisca traffico tramite i protocolli sicuri TLS1.0 e HTTPS

[1] `https://core.telegram.org/bots/webhooks`

– Un certificato autentico o un'autocertificazione

– Utilizzi un CN o SAN che corrispondi al dominio fornito durante l'installazione

– Fornisca tutti i certificati intermedi per completare la catena di verifiche

4.3.2 Long Polling

Non è il client che richiede una particolare risorsa ma è il server che, a fronte di un evento, avvisa il client dell'accaduto. Questo approccio è utilissimo in quelle applicazioni con una forte interazione tra gli utenti (per esempio le chat o le aste in real-time). Il client richiede al server le informazioni come nel normale polling, il server potrebbe non rispondere subito, infatti se dall'ultima chiamata non vi sono stati aggiornamenti il server al posto di inviare una risposta vuota, lascia la domanda aperta e attende che le informazioni di risposta diventino disponibili. Non appena il server riceve le nuove informazioni, le invia al client, il quale dopo avere ricevuto la risposta provvede a rinviare una richiesta di aggiornamenti al server. In questo modo viene eliminata la latenza di risposta che si verifica quando le informazioni sul server sono disponibili ma il client non ha ancora provveduto a formulare la richiesta di aggiornamento. In figura 4.2 è riporatato il diagramma di sequenza delle varie componenti e le loro interazioni. Il metodo di comunicazione tra bot e server Telegram è quello implementato nell'applicazione (Long polling).

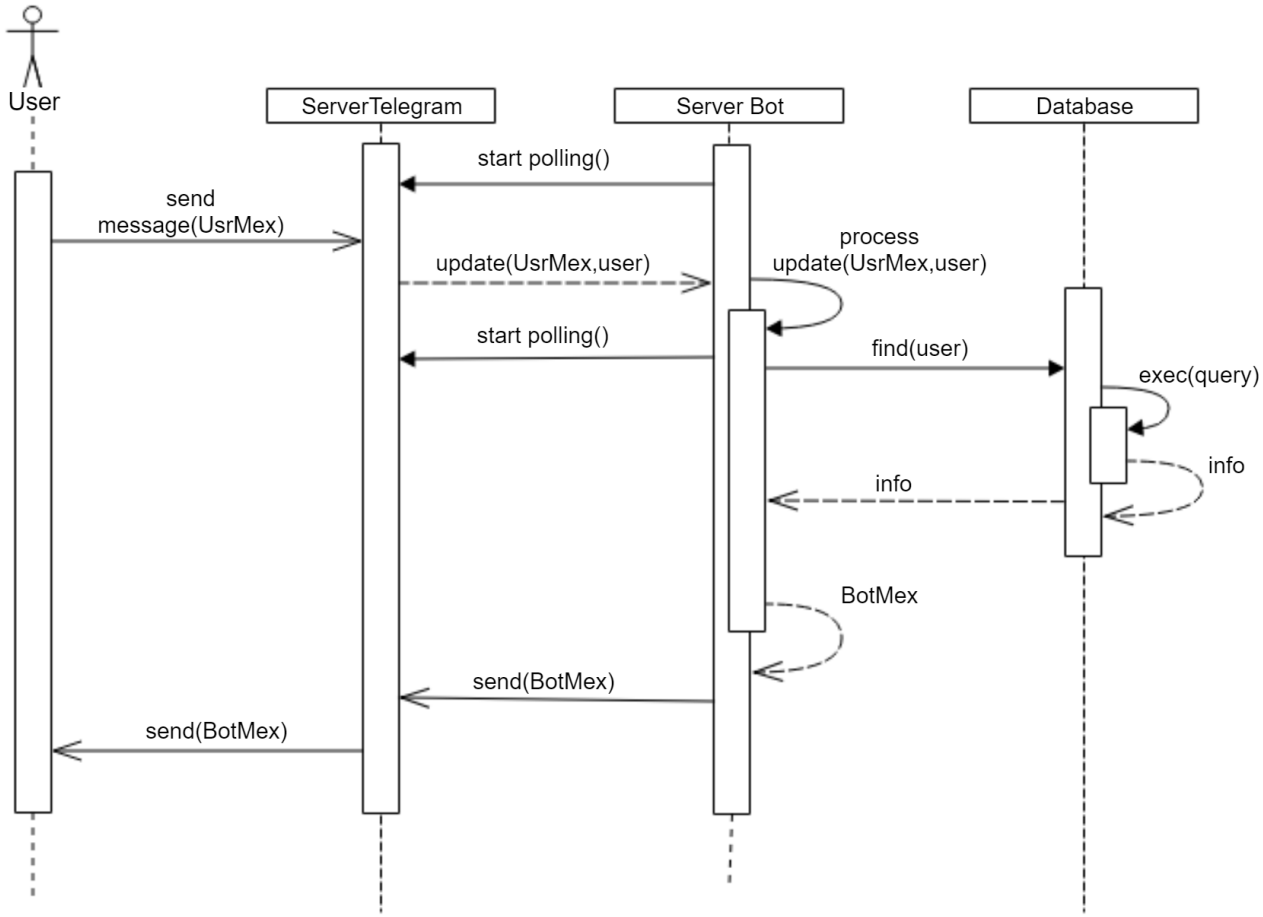

Figura 4.2: Sistema - Diagramma di sequenza

4.4 Architettura del Bot

In fase di scelta su quale architettura fosse più ideale per la realizzazione del server sono state individuate varie possibilità, ma prima è stato fondamentale fare alcune considerazioni e prendere certe decisioni.

4.4.1 Considerazioni sull'uso di AI e NLP

Spesso i chatbot sono sistemi che fanno largo uso di intelligenza artificiale e interpretazione del linguaggio naturale (NLP). Ci si è chiesti se questo fosse strettamente necessario nello sviluppo di un chatbot. Implementazioni che non ne fanno largo utilizzo possono basarsi su una serie di modelli definiti a priori, in questo modo la logica viene codificata 'a mano' all'interno del bot, in questo modo il margine di adattamento a nuovi domini sarà meno flessibile. Il bot in analisi deve soddisfare la specifica dei requisiti definita nel capitolo3, tra questi non è richiesto che ci sia una componente di adattamento a nuovi contesti lavorativi differenti da quello pensato inizialmente. Se la logica del chatbot non è destinata a mutare nel tempo si pensi di implementare l'intero sistema cercando di codificare i vari comportamenti che l'utente potrebbe assumere nell'interrogazione del chatbot. L'architettura individuata che meglio definisce questo comportamento è l'implementazione di una macchina a stati finiti (FSM).

4.4.2 Macchina a Stati Finiti

Una macchina a stati finiti è un modello computazionale utilizzato per rappresentare e controllare un flusso d'esecuzione. È un sistema dinamico, discreto ed invariante, in cui gli insiemi d'ingresso, di uscita e di stato sono finiti. Solo un singolo stato può essere attivo allo stesso tempo, per cui la macchina deve passare da uno stato all'altro per eseguire diverse azioni. Le transizioni fra stati sono ciò che rappresentano l'evoluzione, ovvero il comportamento, di una macchina a stati. Una transizione coinvolge sempre due stati quello di partenza e lo stato di arrivo. Con gli automi a stati finiti, si possono modellare tutti i sistemi che possiedono le seguenti caratteristiche:

– Dinamicità: caratteristica di evolvere nel tempo passando da uno stato ad un altro.

– Discretezza: caratteristica che indica che le variabili d'ingresso e gli stati del sistema da modellare possono essere espressi con valori discreti.

– Simboli finiti: caratteristica che determina che il numero di simboli di ingresso e di stati sia rappresentabile da un numero finito.

Ogni automa a stati finiti è associato univocamente ad un grafo orientato, i nodi del grafo coincidono con gli stati dell'automa. Viene riportato in figura 4.3 il diagramma degli stati del sistema. La macchina a stati è di tipo deterministico. È possibile da ogni stato tornare nello stato Idle tramite l'evento: end.

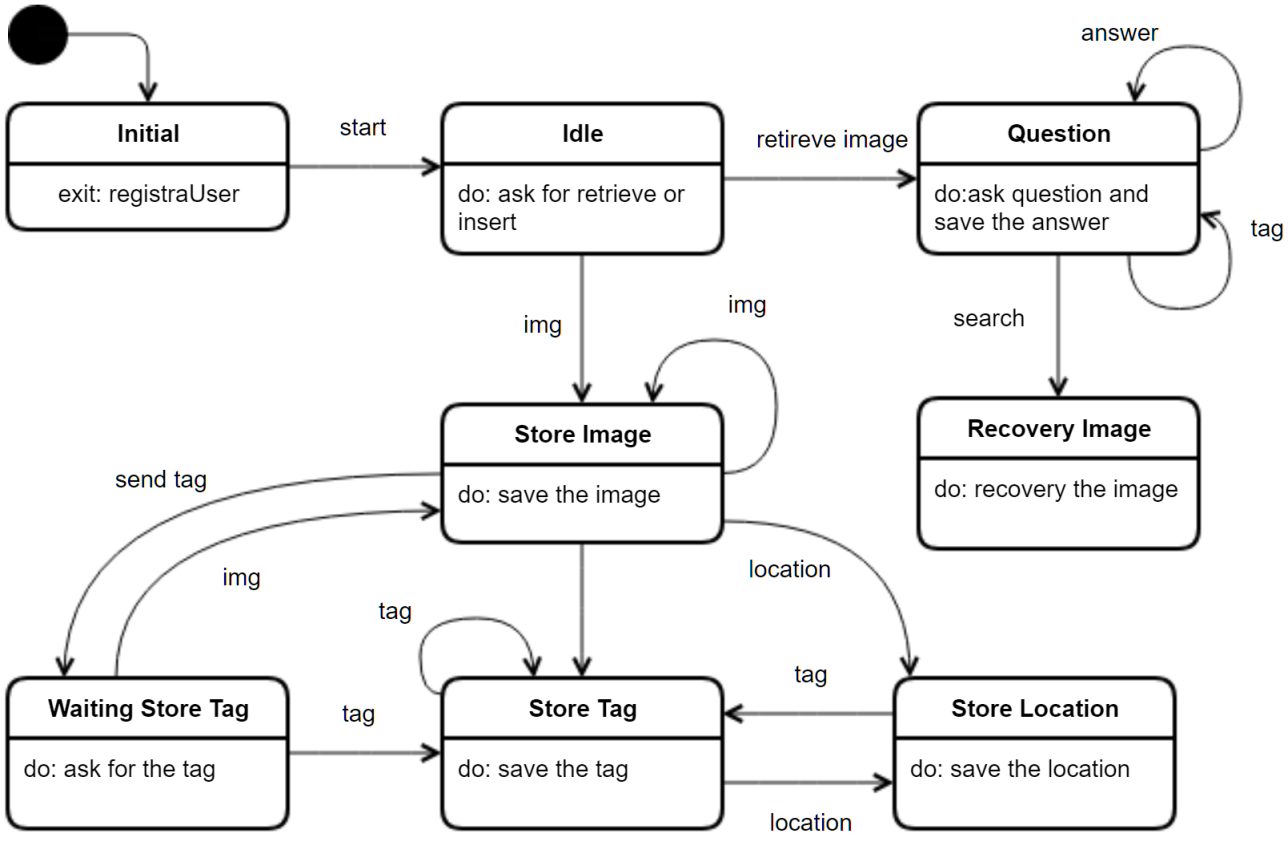

Figura 4.3: Sistema - Diagramma degli stati

4.4.3 Come mantenere lo stato: i bot sono stateless

I bot di default sono stateless, ovvero viene instaurata una connessione tra user e bot in cui ciascun messaggio è indipendente dai precedenti. Sorge un problema di fondo poiché il chatbot che deve essere implementato è un sistema che evolve nel tempo in funzione dell'ordine dei messaggi in ingresso. Se un utente invia un'immagine e nel messaggio successivo una serie di tag, questi sono strettamente correlati alla foto appena inviata. Non si può pensare di trattare il secondo evento(ricezione di tag) senza considerare il primo(ricezione di una foto). La strategia che si è deciso di attuare per risolvere questo problema è stata quella di salvare sul database tutte le modifiche che sono avvenute in ogni stato per un determinato utente. In questo modo, non appena un utente scrive un messaggio al bot, questo recupera dal database il contesto di quell'utente, processa i suoi dati e successivamente salva le modifiche apportate sulla base dati. In questo modo quando l'utente invierà un altro messaggio il bot riuscirà ad avere la storia completa delle azioni passate.

4.4.4 Gestire più utenti

Il sistema riceve le richiesti di più utenti, quindi è necessario che riesca a gestire messaggi provenienti da utenti diversi. Non appena un messaggio viene inviato al bot, si ripristina la storia dell'utente che ha inviato tale messaggio. È possibile recuperarla perché nel database è presente un record per ogni utente dove viene specificato lo stato nel quale si trova e un insieme di informazioni serializzate che dipendono dallo specifico stato di appartenenza. Ogni messaggio in entrata viene gestito in maniera autonoma, è necessario scaricare le informazioni sullo stato di appartenenza dal database. In questo modo 2 messaggi di due utenti diversi che giungono al bot possono essere processati uno dopo l'altro senza andare

a corrompere uno il flusso dati dell'altro e viceversa, questo perché ad ogni messaggio viene ripristinato il contesto relativo.

4.4.5 Diagramma delle classi

Nella programmazione orientata agli oggetti, lo State è un design pattern comportamentale. Esso consente ad un oggetto di cambiare il proprio comportamento a run-time in funzione dello stato in cui si trova. Struttura del pattern:

- Context: Definisce l'interfaccia del client e mantiene un riferimento ad un ConcreteState.
- State: Definisce l'interfaccia, implementata dai ConcreteState, che incapsula la logica del comportamento associato ad un determinato stato.
- ConcreteState: Implementa il comportamento associato ad un particolare stato.

Tra i benefici dell'adozione di questo design pattern vi è il fatto che il comportamento associato ad uno stato dipende solo da una classe (ConcreteState), la logica che implementa il cambiamento di stato viene implementata in una sola classe (Context) piuttosto che con istruzioni condizionali nella classe che implementa il comportamento. In questo modo si evitano stati incoerenti. Tra le conseguenze vi è il fatto di dovere implementare una classe per ogni stato avendo in questo modo un gran numero di classi nel progetto. Per l'implementazione del sistema si è deciso di utilizzare la libreria PyTransitions che fornisce un'implementazione base di una macchina a stati a finiti, si è cercato di sfruttare questa libreria unita ai vantaggi dello State pattern. Segue in figura 4.4 il diagramma delle classi.

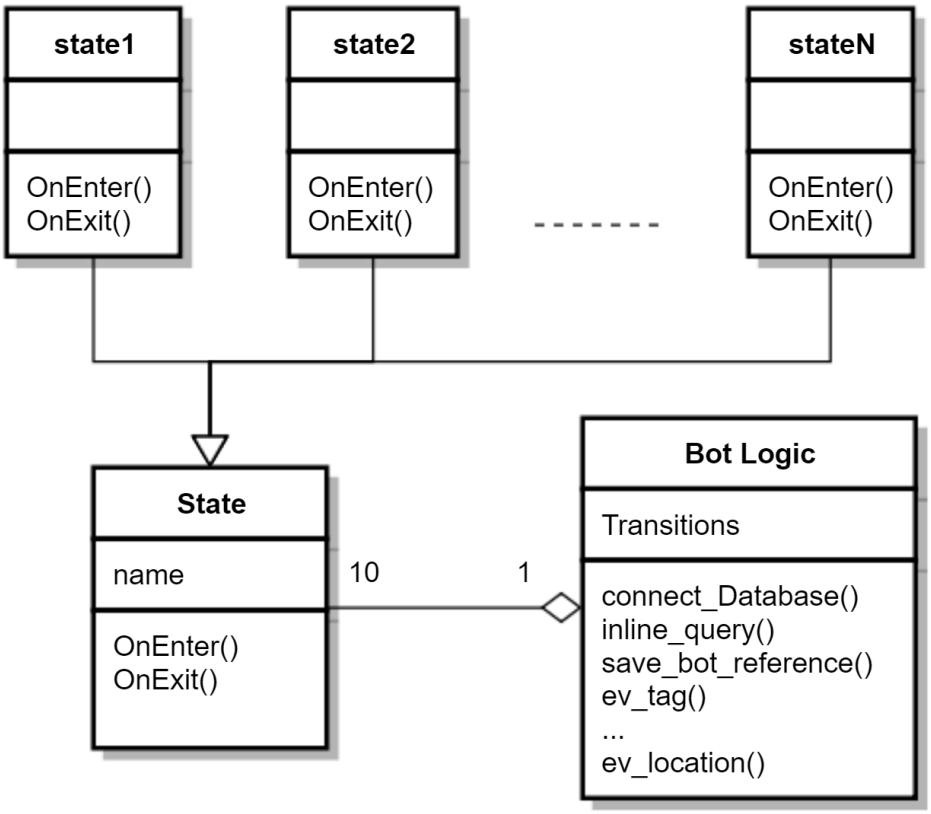

Figura 4.4: Sistema - Diagramma delle classi

4.4.6 Implicit tagging

Il tagging di un'immagine è il processo mediante il quale vengono assegnate una serie di parole chiave all'immagine in questione. Questo compito viene spesso eseguito da persone umane, per questo motivo i tag assegnati potrebbero dipendere dal punto di vista di chi esegue l'operazione. In questa applicazione le immagini rimangono private, ogni utente vede solamente le proprie. Anche se i tag sono assegnati in modo soggettivo, l'importante è che ognuno riesca a recuperare le proprie immagini. Può verificarsi il fatto che una persona non abbia né tempo né voglia di taggare un'immagine. Si potrebbe pensare che questo compito venga demandato al sistema. Lo scopo di questa fase è quello di riuscire a

estrarre il maggior numero di informazioni partendo da una foto. Vi sono svariate strategie e tecniche per estrapolare questi dati. Si potrebbe pensare di estrarre determinate features cercando nei raggruppamenti di pixel dell'immagine. In alternativa si potrebbero utilizzare sistemi e algoritmi che facciano uso di intelligenza artificiale in modo che vadano a ricercare determinati pattern all'interno di una foto. Queste tecniche prevedono di estrarre informazioni partendo dal contenuto informativo dell'immagine. Un' altra tecnica, che sfrutti altri tipi di informazioni, è quella di salvare tipologie di dati non strettamente correlati al contenuto della foto. Questi dati potrebbero essere l'orario, la data o la posizione dove la foto è stata scattata.

Valutando ciò che viene detto in fase di analisi sul dominio applicativo del sistema. È emerso il fatto che il bot potrebbe contenere grandi quantità di fotografie di post-it e bigliettini. Il colore di questi post-it sui quali sono appuntati i dati potrebbe essere una valida informazione da memorizzare. Si pensi che se il post-it ha un colore particolare (verde, giallo, rosso ...) per l'utente potrebbe essere naturale ricordarlo. Tale informazione specificata in fase di ricerca potrebbe essere fondamentale per il recupero dell'immagine. Gli altri dati che possono essere ricavati sono l'orario e la data di invio dell'immagine. Per quanto riguarda la posizione, non si può pensare che esse vanga inviata al bot senza l'autorizzazione dell'utente (per ovvi motivi di privacy). L'unica strategia per avere questa informazione in modo indiretto, potrebbe essere quella di provare ad estrarla dai metadati della foto. Si è deciso che il metodo migliore per ricavare tale dato fosse quello di richiederlo direttamente all'utente. Gli venga chiesto di interagire con un bottone per condividere la propria posizione GPS. Le coordinate dovranno essere trasformate mediante un procedimento di reverse geocoding in un'informazione che riguarda la città o località corrispondente.

Il meccanismo di taggatura appena descritto sarà una delle funzionalità più importanti del sistema. Questo perché permette all'utente di recuperare la propria foto senza doversi pre-

occupare di taggarla opportunamente in fase di inserimento. Il sistema riesce a ricercare la foto richiesta ponendo una serie di domande all'utente. Le domande si riferiscono ai dati che il bot ha ricavato da ogni immagine. Le informazioni richieste potrebbero essere facilmente ricordate dall'utente, in caso contrario può comunicare di non sapere la risposta.

4.4.7 Estrazione colore Post-it

L'operazione di estrazione del colore del post-it deve essere eseguita prima della fase di archiviazione. Si pensi che molte immagini ricevute possano rappresentare foto di appunti annotati su foglietti o etichette colorate. L'estrazione di tale informazione potrebbe essere, per questo motivo, particolarmente interessante. Vanno fatte alcune considerazioni sul tipo di dato che bisogna estrarre. In prima analisi non si dovrà tenere conto di numerose sfumature, basti riconoscere 6/7 colori. L'utente verosimilmente non ricorderà che ha scritto i dati su un foglietto ceruleo o ocra, ma specificherà quelli più comuni. Per questo motivo si è deciso di classificare il dato estratto in uno dei seguenti colori: rosso, arancione, giallo, verde, blu, rosa/viola.
Si è deciso di trovare una libreria che individuasse il colore dominante di una foto e lo restituisse come tupla RGB. La libreria appena descritta è stata identificata in "ColorThief". Per estrarre il colore rappresentato dalla tupla RGB si converti il valore in una terna HSL. Dalla terna appena ottenuta si pensi di definire il colore analizzando la componente H (Hue). A ognuno dei colori elencati in precedenza, è stato assegnato un valore minimo e un valore massimo in modo che venisse rappresentato mediante una range numerico finito. I valori dei range sono definiti all'interno delle 360 tonalità di Hue. Per assegnare ad ogni range un colore, si è adottato un metodo empirico. La decisione su come definire il confine tra il range di un colore e quello del colore successivo, è stata presa mediante l'esperienza diretta del progettista.

La strategia adottata potrebbe non essere la migliore, ma bisogna fare delle considerazioni per le quali potrebbe essere una buona soluzione. La foto che viene processata non è quella originale, ma una foto più piccola e a risoluzione più bassa. Questo comporta che i tempi di elaborazione dell'immagine siano contenuti, inoltre si perde buona parte della qualità della foto. In questo modo non vi sono confini chiari e contrasti netti di colore, piuttosto è presente una macchia di colore uniforme. Si pensi che le scritte sul post-it non siano più nitide e diventi difficile riuscire a separarle dallo sfondo.

4.5 User Experience

Il metodo attraverso il quale il sistema viene utilizzato dall'utente è tramite la chat dell'applicazione di messaggistica Telegram. Per questo motivo si sono studiati quali fossero gli strumenti di interazione con l'utente che questa applicazione fornisse agli sviluppatori di bot. Sono stati individuati 4 elementi fondamentali che il bot potesse utilizzare:

- Messaggi di testo e invio di foto: Il metodo più immediato e semplice per comunicare delle informazioni in una chat è l'invio di un messaggio testuale. Quando il bot recupera l'immagine che l'utente ha richiesto questa viene inviata come messaggio all'interno della chat. Non sono richieste conoscenze particolari che non siano quelle di scrittura e invio di un messaggio e di una foto, operazioni note all'interno di una chat.

- InlineKeyboardButton: bottoni all'interno della chat. Il bottone è uno dei metodi più efficaci per permettere all'utente di prendere una decisione a fronte di varie opzioni.

- KeyboardButton: bottoni fissi sulla tastiera. Corrispondono a comandi.

- Comando: seguendo la sintassi ‘ /nomeComando ‘ è possibile specificare comandi al bot. I nomi dei comandi devono essere ricordati e bisogna seguire la sintassi corretta.

- Emoticon: Faccine stilizzate o particolari immagini di piccole dimensioni che hanno un elevato contenuto informativo poichè idealizzano un concetto o comportamento, con un disegno.

Si è deciso di non utilizzare i comandi in quanto è richiesto che gli utenti li imparino a memoria e utilizzino una sintassi particolare, rendendo l'uso del bot più complicato per coloro che non hanno mai interagito con uno di questi. Si decida di utilizzare le emoticon per rafforzare il significato che del testo scritto vuole comunicare. Spesso infatti un particolare simbolo posto in un determinato contesto, può essere molto più espressivo di un messaggio testuale che specifichi una qualche direttiva. Si pensi di utilizzare tale strumento per rendere i messaggi più comprensibili e gradevoli alla vista. È specifica dei requisiti che l'utente non debba avere conoscenze che vanno oltre l'uso di una chat, per questo motivo si è deciso di implementare il sistema sfruttando i messaggi e i bottoni. I bottoni in realtà non si trovano comunemente in applicazioni di messaggistica e nelle chat, però considerandoli come uno strumento estremamente intuibile e la cui conoscenza dovrebbe già essere nota ad un utente che sia in grado di utilizzare uno smartphone, si è preferito adottarli.
Le operazioni sono veloci perché permettono di concludere l'operazione di inserimento con 2 soli messaggi e pochi di più per le operazioni di ricerca e rimozione. Le conoscenze utilizzate sono quelle relative all'uso di una chat, con l'eccezione dei bottoni. Per valutare se le decisioni prese soddisfassero i requisiti di usabilità è stata prodotta una versione minimale del sistema dove ci si è limitati a implementare solamente le parti relative all'interazione con l'utente. Dopo aver fatto testare l'applicazione a diverse persone si è tenuto conto delle varie considerazioni e consigli.

4.5.1 Layout nelle operazione di inserimento

Per le operazioni di inserimento si è deciso di ridurre al minimo le interazioni con il sistema e di rendere questa operazione il più immediato possibile. Si è pensato di far comporre un messaggio per l'invio della foto seguito da uno per l'invio dei tag correlati o un click su un bottone nella tastiera per l'invio della posizione.

4.5.2 Layout nelle operazioni di ricerca

Nelle operazioni di ricerca l'utente può essere indotto a digitare qualche messaggio in più rispetto la fase di inserimento perché mosso dall'interesse di recuperare l'informazione richiesta. Difficilmente farebbe a meno di utilizzare il bot. Questa operazione può essere più o meno articolata in base al tipo di ricerca che deve essere effettuata. Si dovrà comunicare l'intenzione di volere recuperare un'immagine cliccando su un bottone; successivamente basterà inserire i tag e cliccare sul bottone di ricerca. Se si vuole effettuare una ricerca più raffinata il sistema pone delle domande all'utente che può rispondere tramite 3 bottoni: Si - No - Non so. L'operazione di rimozione può essere effettuata dopo una funzione di ricerca interagendo con un bottone.

4.6 Sicurezza e crittografia

La comunicazione tra l'utente e il bot passa attraverso i server di Telegram, per rendere la comunicazione sicura è implementato il protocollo MTProto[2]. La crittografia [3] è basata sulla crittografia AES simmetrica a 256-bit, sulla crittografia RSA 2048 e sullo scambio di chiavi di sicurezza Diffie-Hellman. Telegram è aperto e chiunque può controllare il codice sorgente, il protocollo e le API. Questo non vuol dire che Telegram sia sicuro al 100%

[2] `https://core.telegram.org/mtproto`
[3] `https://telegram.org/faq/it#d-quanto-sicuro-telegram`

ma quanto basta per permetterci di fidarci di tale applicazione. Al fine di utilizzare tale applicazione per la memorizzazione di dati sensibili sarebbe inoltre necessaria una forma di crittografia prima del salvataggio su database. Tuttavia, tale caso d'uso esula da quanto sviluppato in questa tesi e pertanto si demanda a sviluppi futuri.

4.7 Architettura della base dati

La base dati del chatbot deve gestire degli utenti e per ognuno di essi è necessario che si tenga nota dello stato nel quale si trova. Ad ogni stato corrispondono una serie di informazioni variabili che mutano in funzione delle scelte effettuate dall'utente. Queste si ripercuotono nei comportamenti degli stati successivi e quindi è necessario tenere nota delle modifiche dei parametri interni a ogni stato. Un utente è identificato da un chatID univoco e può possedere una serie di immagini. Le immagini sono identificate da un codice univoco all'interno di Telegram, per ognuna di esse si memorizzi la data di inserimento. Per ogni immagine si possono memorizzare anche il luogo di scatto e se si tratta di un post-it, il colore del foglietto. Ad ogni immagine inoltre possono essere associati un numero variabile di parole chiave.

4.7.1 Considerazioni preliminari

Le immagini vengono memorizzate all'interno dei server di Telegram ed è possibile recuperarle tramite un codice univoco (FileID). Per tale ragione si è deciso di non scaricare tutte le immagini che l'utente invia al bot per poi doverle caricare nella propria base dati. Si è deciso di utilizzare i fileID univoci come identificativi per ogni immagine, e quando nel momento del bisogno è necessario inviare all'utente la foto richiesta, grazie al fileID la si identifichi sul server Telegram e la si faccia inviare all'utente. Questa scelta potrebbe non

essere ideale nell'ottica di volere mantenere una copia delle immagini anche nella propria base dati, ma è nota d'mportanza il fatto che in questo modo si alleggerisce il traffico dati in entrata al bot. Un'immagine pesa molto di meno in termini di memoria rispetto un codice a 64 byte, ne beneficiano le performance e le specifiche dell'hardware e di rete necessari al funzionamento del sistema. Basti pensare che in un'applicazione del genere ogni interrogazione effettuata dall'utente al sistema, richiede l'invio o la ricezione di una foto. Considerando che quanto appena detto è valido per ognuno dei numerosi utenti che utilizza l'applicazione nello stesso arco temporale, ci si rende conto dell'enorme differenza tra l'invio di una moltitudine di stringhe o di immagini. Il traffico dati oneroso viene demandato al server di Telegram, che dovrebbe svolgere tale compito ugualmente anche nel caso che le immagini vengano memorizzate sul server del bot, questo perché fa da nodo intermediario tra l'utente e il server del bot. Telegram stesso suggerisce indirettamente questa soluzione poiché fornisce API particolarmente rigide e poco flessibili per operazioni di download e upload di immagini.

4.7.2 Schema concettuale

Partendo dalla descrizione del problema è stato prodotto lo schema concettuale in figura 4.5 mediante rappresentazione come modello entity-relationship:

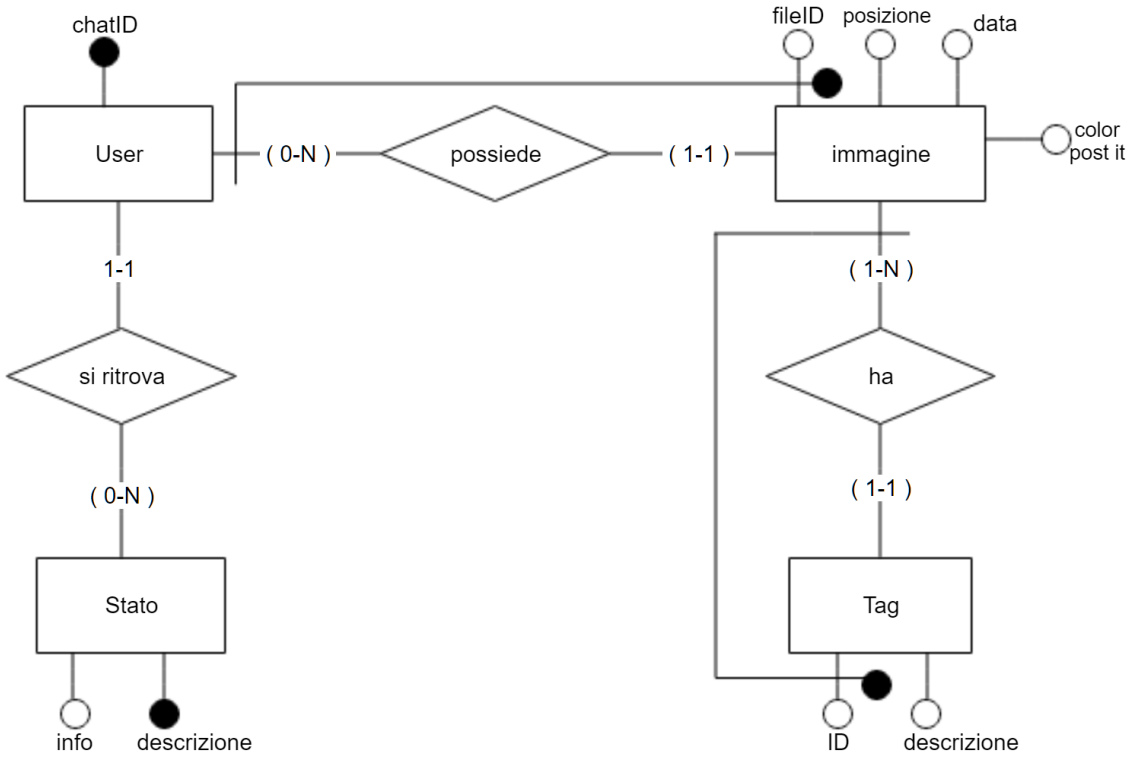

Figura 4.5: Sistema - Modello Entity-Relationship

Segue la tabella 4.1 dove vengono descritte le associazioni e relazioni.

Nome	Tipo	Descrizione
User	E	Rappresenta ogni utente che utilizza il chatbot
Immagine	E	Rappresenta l'immagine salvata da un utente
Stato	E	Rappresenta lo stato (della FSM) nel quale si trova l'utente
Tag	E	Parola-chiave riferita a un'immagine
possiede	R	Lega un Utente alle immagini che ha inviato al chatbot
ha	R	Lega un'immagine alle parole chiave ad essa correlate
Si trova	R	Lega un utente allo stato nel quale si trova

Tabella 4.1: Tabella Descrizione entità e associazioni

4.7.3 Operazioni principali

- Op1: Inserimento di un nuovo utente.
- Op2: Inserimento di un'immagine.
- Op3: Aggiunta di tag.
- Op4: Aggiunta della posizione.
- Op5: Ricerca di un'immagine.
- Op6: Ricerca di un'immagine forniti i tag.
- Op7: Ricerca di un'immagine con informazioni ibride.

4.7.4 Scelta degli identifictori primari

Ogni entità deve essere identificata da informazioni univoche in modo di evitare che si verifichino situazioni di ambiguità.

User: La chat di ogni utente viene identificata all'interno di Telegram con un codice univoco, si utilizzi questo valore per identificare ogni utente.

User: ChatID

Stato: Il nome di ogni stato.

Stato: nome

Immagine: Ogni immagine viene identificata da un codice univoco (assegnato da Telegram) e dall'utente che ha inviato la foto.

Immagine: FileID,ChatID

Tag: Identificato da un codice univoco.

Tag: ID

4.7.5 Schema relazionale

Si è deciso di importare in User le informazioni relative allo Stato nel quale si trova. Tutte le operazioni che richiedono al bot di conoscere con quale utente sta intrattenendo una conversazione, richiedono anche la conoscenza dello stato di appartenenza dell'utente in quel preciso momento.
Le interrogazioni relative all'inserimento/ricerca di un'immagine sono sempre correlate a un preciso utente, si è deciso quindi di importarne un riferimento in Immagine.
Ogni tag è identificato da un codice univoco, ma è anche correlato a un'immagine che appartiene a un utente. Le operazioni di ricerca in base ai tag si stimi che siano frequenti, per tale motivo si è deciso di importare un riferimento all'immagine e all'utente all'interno di Tag.

User (ChaID, Stato: Nome, Stato: Info)

Immagine (FileID, User: ChatID, Date, Location, Color, Lastview)

Tag (Id, User: ChatID, Immagine: FileID , Tag)

Stato (Nome, Info)

Capitolo 5

Realizzazione del sistema

5.1 Creare un chatbot telegram

Per creare un bot su telegram si contatti il BotFather, è il bot che gestisce tutti i chatbot creati da terze parti. Tramite questo bot se ne possono creare di nuovi o modificare le impostazioni di quelli esistenti. Con il comando /newbot se ne crei un nuovo, dopo avergli assegnato un nome, viene generato un token di autorizzazione per il bot appena creato. Il token è una stringa che definisce univocamente il bot ed è necessario per l'invio delle richieste alle API.

Figura 5.1: Immagine - Botfather

5.2 Ricevere gli aggiornamenti

Viene creato un oggetto 'Updater' che si occupa di ricevere tutti gli aggiornamenti in arrivo al bot dal server di telegram, questi dati vengono trasmessi dall'oggetto updater a un 'Dispatcher' al quale vengono agganciati una serie di handler che si occupereranno di gestire nel modo più opportuno i vari eventi in entrata. Per implementare il servizio di ricezione dei messaggi tramite il processo a LongPolling, basti invocare il metodo: 'start_polling()' sull'oggetto che si occupa di riceve gli aggiornamenti (Updater). Dopo che si è ricevuta una risposta, è possibile decidere il tempo che si vuole attendere prima di inviarne una nuova. Di default questo intervallo temporale è impostato a 0 secondi per evitare che vi siano nuovi aggiornamenti che non vengono richiesti.

5.3 Tipologie di handler

È possibile connettere al Dispatcher una serie di handler. Il dispatcher riceve un evento ed esegue l'handler corrispondente grazie a un sistema di CallBack. L'utente può comunicare con il bot tramite: l'invio di comandi, la pressione di un bottone o l'invio di messaggi normali (possono contenere testo, posizione GPS, foto). Vi sono 3 tipi di handler differenti, uno per ogni casistica.

Ricezione di un comando Ogni utente può inviare un comando al bot scrivendo nella chat un messaggio che segua la seguente sintassi: ' /nomeComando ' 'eventuale testo'. Per catturare eventi di questo tipo si utilizzi un CommandHandler, sarà necessario specificare il nome del comando e il metodo da invocare per gestire l'evento. Viene utilizzato per dare l'invio del comando 'start' la prima volta che si utilizza il sistema.

Ricezione di un messaggio testuale i messaggi normali vengono catturati tramite un MessageHandler, è necessario specificare il contenuto del messaggio tramite un filtro

per verificare che si tratti di testo, di una posizione o di una foto. Per ogni tipologia di contenuto differente vi sarà il corrispondente metodo di callback.

Pressione di un bottone In telegram i bottoni possono essere di due tipi: sulla tastiera o nella chat. I primi sono i Keyboardbutton, in realtà corrispondono all'invio di un comando. Sono un metodo più semplice per far impartire un comando senza obbligare l'utente a ricordare la sintassi e il nome specifico. Viene scritto del testo più comprensibile per l'utente sul bottone e se viene premuto viene inviato il comando corrispondente che viene gestito dal CommandHandler come appena specificato. Gli altri tipi di bottoni, sono sullo schermo e sono gli InlineQueryButton. L'entità che cattura l'evento di pressione di uno di questi bottoni è il CallbackQueryHandler. Anche in questo caso bisogna specificare il nome del metodo da invocare come callback.

è riportato un esempio di codice dei principali tipi di handler che vengono utilizzati:

```
1 dispatcher.add_handler(CommandHandler('start', comando_start ) )
2 dispatcher.add_handler(MessageHandler(Filters.text, ricezione_tag ) )
3 dispatcher.add_handler(MessageHandler(Filters.photo, ricezione_foto ) )
4 dispatcher.add_handler(CallbackQueryHandler(inline_query ) )
```

I vari Handlers tramite una callback invocheranno i metodi: comando_start/ricezione _tag/ricezione_foto... dove si avrà una strategia per gestire tutti gli eventi all'interno della macchina a stati finiti.

5.4 Logica del bot

BotLogic è la classe principale del bot dove viene gestita tutta la logica che governa la macchina a stati. La libreria 'transitions' fornisce una buona astrazione per implementare la macchina a stati. Definiamo le transizioni tramite una lista di dizionari (struttura dati propria di python) in questo modo si specifica l'evento che fa scattare la transizione, lo stato di partenza e lo stato di arrivo. Si possono anche specificare transizioni che avvengono solo dopo aver valutato un insieme di valori. Dentro la classe che gestisce la logica vengono istanziati gli stati della macchina e instaurata la connessione con il database MySQL. L'esecuzione dei vari handler, descritti nel paragrafo 5.3, viene gestita in questa classe. L'arrivo di un aggiornamento al bot determina che venga invocata la corrispondente funzione di callback, tale metodo viene definito all'interno di questa classe. In funzione della natura del messaggio ricevuto, viene creato l'evento corrispondente e ,se risulta significativo per lo stato nel quale la macchina si trova, scatterà la transizione determinando l'uscita dallo stato.

5.5 Stati

Si noti che la logica di funzionamento e coordinazione tra i vari utenti e processi è incapsulata in BotLogic. Le funzionalità vere e proprie che permettono al bot di svolgere il compito per il quale è progettato, vengono implementate in ogni singolo stato. Ogni stato eredita dalla classe State, questa classe fornisce un'impalcatura generale per ogni stato del sistema. È possibile definire un insieme di callback in fase di entrata e uscita da uno stato.

- OnEnter: Handler eseguito tutte le volte che si entra in uno stato.
- OnExit: Handler eseguito tutte le volte che si esce da uno stato.

L'ultima operazione che è necessario eseguire prima di tornare in attesa di nuovi eventi, è quella di salvare nel database lo stato corrente e gli aggiornamenti avvenuti. In questo modo quando il medesimo utente comunicherà nuovamente con il bot, i dati aggiornati verranno scaricati e viene riproposto lo stesso ambito. Questa operazione di download dello stato appena un utente contatta il bot, e upload dello stato non appena il bot ha eseguito una serie di operazioni, servono per mantenere lo stato della conversazione, poiché i chatbot di default sono stateless.

5.6 Transizioni

Le transizioni potevano essere implementate nello stesso modo degli stati, si poteva avere un oggetto per ogni transizione definendo lo stato di partenza, lo stato di arrivo ed eventuali funzioni che definissero il comportamento in fase di transizione tra stati. Non essendoci comportamenti particolari per ogni transizione, si è deciso di avere un codice pulito evitando di avere una classe per ogni transizione (come si è invece fatto per gli stati). Le transizioni sono dichiarate dentro BotLogic come una lista di dizionari. Viene specificato il nome dell'evento che fa scattare la transizione, lo stato di partenza e quello di arrivo. Le transizioni vengono definite come di seguito:

```
transitions = [
      {'trigger': 'start','source':'initial','dest':'idle'},
      {'trigger': 'img','source':'idle','dest':'store_image'},
      {'trigger': 'search','source':'question','dest':'recovery_image'},
      {'trigger': 'location','source':'store_image','dest':'store_location'}
]
```

Il valore corrispondente a ‘ trigger ‘ è il nome dell’evento. Il valore associato ‘ source ‘ è il nome dello stato di partenza, mentre quello associato a ‘ dest ‘ è il nome di quello di arrivo. Ad esempio, se l’utente XYZ si trova nello stato ‘question’ e si presenta l’evento ‘search’ allora l’utente esce dallo stato di appartenenza ed entra in ‘ recovery_image’.

5.7 Eventi

Le transizioni avvengono solamente se si verifica l’evento che le determina. Vi è la creazione di un evento tutte le volte che arrivano degli aggiornamenti al bot, questo non significa che vi deve essere anche un cambiato lo stato. Se l’utente causa un evento che, considerato lo stato di appartenenza, non è correlato a nessuna transizione, tale evento viene ignorato dalla macchina a stati. Possono verificarsi i seguenti eventi:

- Start: Un utente non registrato vuole iniziare a dialogare con il bot.
- Img: L’utente ha inviato un’immagine al bot.
- Send_tag: Intenzione dell’utente di volere inviare una serie di tag.
- Tag: L’utente ha inviato un tag al bot.
- Location: L’utente ha inviato la sua posizione.
- End: L’utente vuole terminare la navigazione e tornare allo stato iniziale.
- Retrieve_image: Intenzione dell’utente di volere ricercare una propria immagine.
- Search: L’utente fa partire la ricerca di un’immagine a partire dai dati forniti.
- Answer: L’utente ha digitato una risposta a una domanda posta dal bot.

5.8 Implementazione delle funzionalità

5.8.1 Inserimento di un'immagine con tag

Gli stati che determinano l'inserimento di un'immagine sono:

– StoreImage: L'utente ha inviato una foto, se non è già presente nel database essa viene memorizzata. Successivamente si chiede all'utente se ha intenzione di fornire la posizione, in modo da connettere l'immagine appena inserita con il luogo dove è stata scattata. Si può inoltre inserire una serie di tag correlati all'immagine per rendere la fase di ricerca più immediata. L'utente ha la possibilità di inviare la foto senza specificare né posizione né tag correlati. Gli altri dati che vengono memorizzati sono la data, l'ora e il colore del post-it.

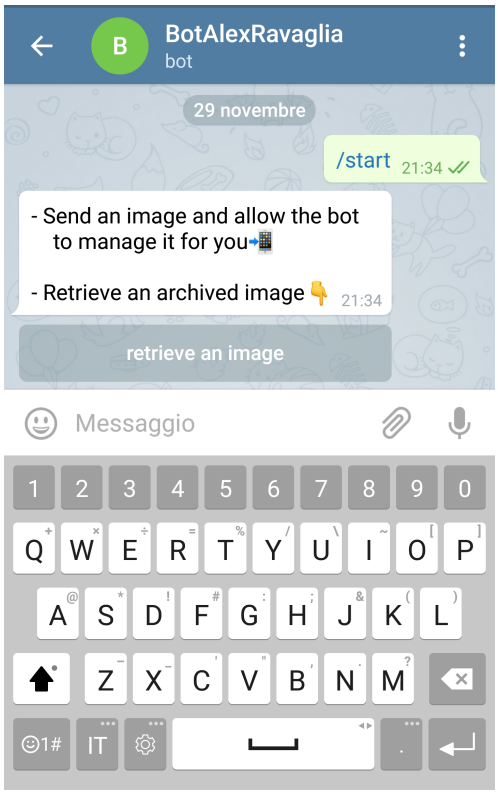

Figura 5.2: Immagine - Bot in attesa di un'immagine

Per l'estrazione del colore del post-it si implementi la strategia espressa in fase di design, viene riportato il codice della funzione che converte una tupla RGB in una HSV.

```
photo = update.message.photo[-1].file_id
date = datetime.datetime.now().strftime('\%Y-\%m-\%d \%H:\%M:\%S')
sql = "INSERT INTO immagine(immagine,user,date) VALUES(\%s,\%s,\%s)"
val = (photo,user, date)
mycursor.execute(sql,val)
mydb.commit();
self.save_state()

def rgb_to_hsv(self,t):
    r, g, b = t[0]/255.0, t[1]/255.0, t[2]/255.0
    mx = max(r, g, b)
    mn = min(r, g, b)
    df = mx-mn
    if mx == mn:
        h = 0
    elif mx == r:
        h = (60 * ((g-b)/df) + 360) \% 360
    elif mx == g:
        h = (60 * ((b-r)/df) + 120) \% 360
    elif mx == b:
        h = (60 * ((r-g)/df) + 240) \% 360
    if mx == 0:
        s = 0
    else:
        s = df/mx
```

```
26      v = mx
27      return (h, s, v)
```

– StoreLocation: vengono aggiornati i dati relativi alla posizione dell'immagine appena inserita. All'utente viene chiesto di condividere le proprie coordinate GPS con il bot. I dati che vengono memorizzati nel database non sono le coordinate, ma il nome della città nella quale l'utente si trova. Per effettuare la conversione da coordinate GPS a indirizzo fisico si utilizzi il servizio di reverse geocoding messo a disposizione da Mapbox.

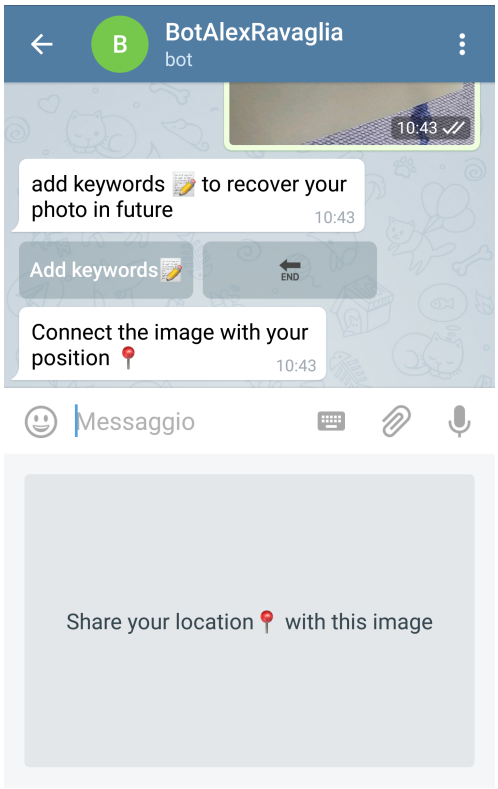

Figura 5.3: Immagine - Aggiungere la posizione o tag

Segue, in figura 5.4, il diagramma di sequenza che mostri le interazioni tra le principali entità in gioco. Per mostrare un diagramma più snello e specifico si è omesso il server

di Telegram, esso si pone da intermediario tra l'utente e il server del bot.

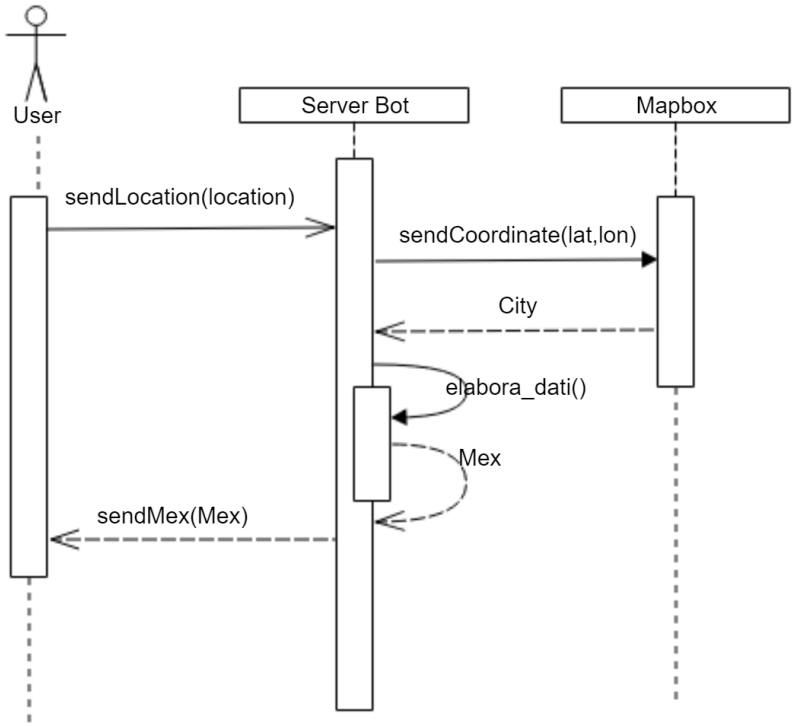

Figura 5.4: Sistema - Diagramma di sequenza tra User - Bot - Mapbox

segue il codice relativo alle operazioni di reverse geocoding.

```
mapbox_token = 'pk.....X9z0A'
.
lon= info['store']['lon']
lat= info['store']['lat']
response = self.geocoder.reverse(lon=lon, lat=lat)
coll=response.geojson()
location = coll['features'][0]['context'][1]['text']
photo = info['store']['photo']
```

```
sql = "UPDATE immagine SET location ='"+str(location)+"' WHERE user='"+str(
    user)+"' AND immagine ='"+str(photo)+"'"
mycursor.execute(sql)
mydb.commit()

end = InlineKeyboardButton("end", callback_data='end')
keyboard = [[end]]
kb_markup = InlineKeyboardMarkup(keyboard)
bot.send_message(chat_id=user , text='Please Insert one keyword at a time',
    reply_markup=kb_markup)

self.save_state()
```

- StoreTag: memorizza una serie di tag digitati dall'utente riferiti all'immagine appena inserita. Questa fase non è strettamente necessaria al fine di memorizzare l'immagine, dovrà essere possibile ripristinarla anche senza aver fornito dei tag.

```
tag = self.m.update.message.text.lower()
photo = info['store']['photo']
sql = "INSERT INTO tag(user,immagine,tag) VALUES(\%s,\%s,\%s)"
val = (self.m.user, photo , tag)
mycursor.execute(sql,val)
mydb.commit();

end = InlineKeyboardButton("end", callback_data='end')
keyboard = [[end]]
kb_markup = InlineKeyboardMarkup(keyboard)
bot.send_message(chat_id=self.m.user , text='add another keyword or end',
    reply_markup=kb_markup)
```

```
12
13 self.save_state()
```

5.8.2 Ricerca di un'immagine

La ricerca di un'immagine è tra le funzionalità più importanti del sistema. Si può effettuare una ricerca tramite tag o rispondendo a una serie di domande poste dal sistema. Si pensi che nel caso si effettui una ricerca per tag, vi è la possibilità di raffinare tale ricerca con la stessa strategia usata per recuperare le immagini senza tag.

Tramite Tag

È il metodo più veloce, è necessario che l'utente specifichi almeno una delle parole chiave con le quali ha memorizzato l'immagine. Il sistema richiede all'utente di inserire i tag che ricorda, successivamente viene effettuata una ricerca nel database. Le immagini trovate sono quelle che sono correlate a tutti i tag digitati. L'immagine o le immagini che soddisfano la ricerca, nel caso non siano più di 5, vengono inviate all'utente che le ha richieste. Se le immagini trovate sono in numero elevato, ciò è sintomo del fatto che l'utente ha salvato molte immagini con gli stessi tag. In questo caso è possibile raffinare la ricerca.

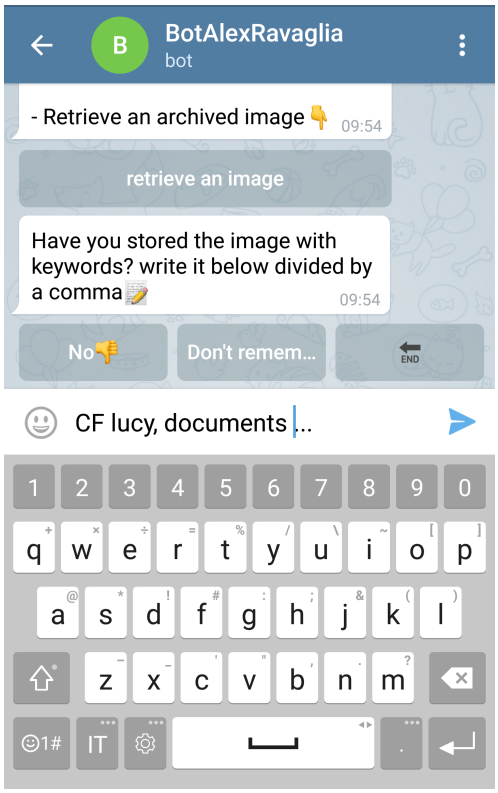

Figura 5.5: Immagine - Ricercare la foto tramite tag

Senza Tag & Raffinamento della ricerca

L'utente non ha inserito nessun tag, o ne ha inseriti pochi in modo che la ricerca non sia risultata efficiente. A questo punto il sistema inizia a porre una serie di domande all'utente per cercare di capire quale sia l'immagine che deve essere recuperata. Il sistema chiede una serie di informazioni basandosi sui dati ricavati autonomamente. Verranno richieste informazioni quali il momento della giornata che è stata scattata la foto, quanto tempo è passato da tale momento, la città e il colore del post-it. Partendo da queste informazioni chieste all'utente, il sistema stesso riesce ad avere abbastanza informazioni per recuperare le foto.

Di seguito in figura 5.6 e 5.7 gli screen delle domande per effettuare la ricerca sui dati di posizione e colore del post-it.

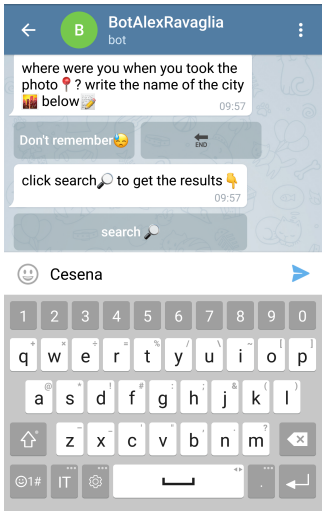

Figura 5.6: Immagine - Ricerca tramite la posizione

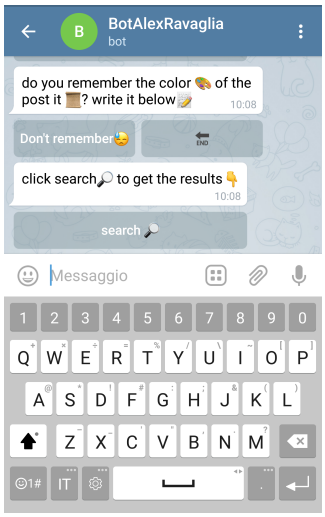

Figura 5.7: Immagine - Ricerca tramite colore del post-it

di seguito viene riportato il codice.

```
TAG="Have you stored the image with keywords? write it below divided by a comma"
LOCATION = "where were you when you took the photo? write it below"
DATE ="when did you send the photo?"
HOUR = "what moment of the day?"
COLOR_POST_IT = "do you remember the color of the post it? write it below"

save_answer()

ask_location()
...
ask_color_post_it()

search_b = InlineKeyboardButton("search", callback_data='search')
keyboard = [[search_b]]
kb_markup = InlineKeyboardMarkup(keyboard)
bot.send_message(chat_id=user, text= "click search button below", reply_markup=kb_markup)

save_state()
....
def ask_location(self):
    info['recovery']['just_requested']=Request.location.name

    b_no = InlineKeyboardButton("no", callback_data='no')
    b_dont_remember = InlineKeyboardButton("Don't remember", callback_data='no')
    b_end = InlineKeyboardButton("end",callback_data='end')
```

```
26      keyboard = [[b_no],[b_dont_remember],[b_end]]
27      kb_markup = InlineKeyboardMarkup(keyboard)
28      send_message(chat_id=self.m.user, text= Question.LOCATION, reply_markup=
    kb_markup)
```

Le operazioni di ricerca restutiscono la foto che corrisponde ai dati forniti. Se l'utente ha risposto in maniera errata alla domande poste dal bot, la ricerca della foto avrà esito negativo.

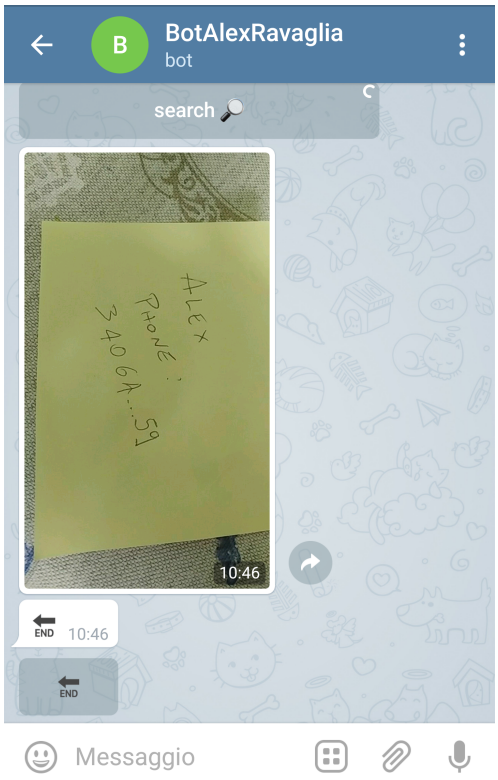

Figura 5.8: Immagine - Ricerca eseguita con successo

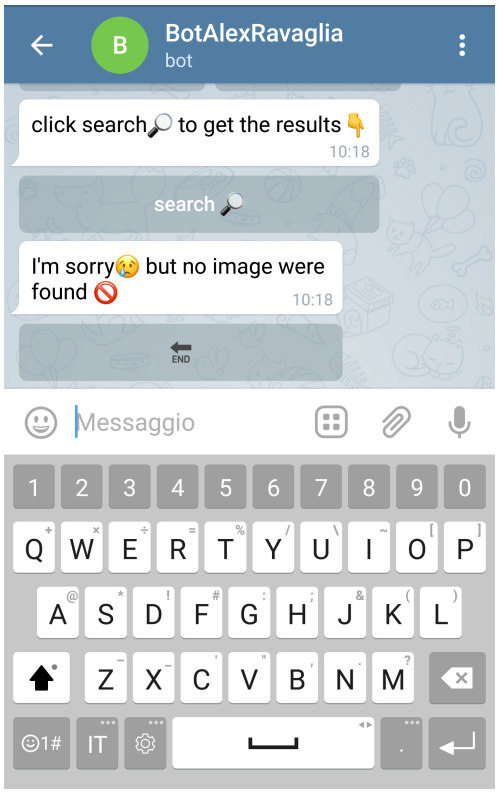

Figura 5.9: Immagine - Ricerca fallita

5.8.3 Rimozione di un'immagine

L'operazione di rimozione viene eseguita in seguito a un'operazione di ricerca, l'immagine può essere eliminata agendo su un bottone sullo schermo.

Capitolo 6

Testing

È impossibile verificare se il sistema che è stato implementato esegue correttamente e non vi sono errori. Per cercare di ridurre al minimo la probabilità di comportamenti inaspettati, sono stati eseguiti una serie di test sulle funzionalità principali e le sezioni più critiche. Partendo da casi noti e conoscendo il risultato che dovrebbe essere prodotto in seguito al susseguirsi di un insieme di azioni specifiche, si testi se le operazioni svolte dal sistema producono quanto aspettato.

6.1 Macchina a stati

Ponendo la macchina a stati in diversi scenari si è simulato in modo automatico, l'arrivo di ogni tipologia di evento e si è testato se la macchina a stati fosse "robusta" e non ci fossero casi di transizioni e comportamenti inaspettati.

6.2 Inserimento di un'immagine nel database

Il sistema non viene utilizzato da un utente alla volta, quindi in fase di inserimento di un'immagine da più utenti, potrebbe succedere che uno di questi sia in grado di memorizzare tra le proprie immagini, foto appartenenti ad altri, o memorizzare i propri tag su immagini non sue. Questi test sono stati eseguiti simulando un traffico in entrata proveniente da più utenti in contemporanea che eseguissero tutti operazioni di inserimento di foto, aggiunta di tag e della posizione. Successivamente sono stati scaricati i dati relativi a ogni utente dal database e si è verificato se fossero conformi con le operazioni che erano state eseguite in precedenza.

6.3 Ricerca di un'immagine

Nella fase di ricerca i principali test eseguiti sono stati di due tipi:

- Ogni utente deve riuscire a recuperare solamente le proprie foto: è situazione di errore che si riescano a scaricare foto appartenenti ad altri utenti, il principale motivo per il quale si può verificare tale problema è uno scenario analogo a quello illustrato nel paragrafo 6.2. Sono state effettuate da più utenti ricerche in contemporanea e si sono verificate le risposte.

- Analisi della correttezza della ricerca: In fase preliminare si è testato se le query venissero create in modo conforme con le informazioni fornite dall'utente, successivamente sono stati eseguiti una serie di test per valutare se le query di ricerca restituissero i risultati aspettati. Si sono prodotte query di tutte le tipologie cercando di riprodurre il maggior numero possibile di combinazioni di dati richiesti.

6.4 Geocoding

Sono state prese una serie di coordinate appartenenti a diverse città e si è testato se la conversione delle coordinate GPS in nome della città di appartenenza fosse corretta.

6.5 Usabilità

L'applicazione è stata fatta testare a diverse utenti, alcuni di questi, coloro che possedevano maggiori praticità con l'uso dello smartphone, hanno utilizzato il bot correttamente e ne hanno imparato le dinamiche in poco tempo. Gli utenti meno pratici invece hanno impiegato qualche momento in più e commesso qualche errore, ma nel complesso hanno riferito quanto il sistema fosse chiaro e intuitivo.

Capitolo 7

Conclusioni

Il sistema implementato soddisfa i requisiti presentati in fase di analisi ed è stato creato per risolvere un problema pratico. Il sistema è fatto in modo che possa essere utilizzato sia da utenti esperti, sia da quelli che possiedono meno dimestichezza nell'utilizzo dello smartphone. Il chatbot creato permette mediante l'uso di una chat su Telegram di inviare una serie di immagini e fare in modo che queste vengano elaborate e salvate. L'utente potrà corredare ogni immagine con una serie di tag per rendere la ricerca più agevole. In alternativa il sistema permette che l'utente invii solamente una foto, in questo scenario è necessario che sia il sistema a taggare autonomamente l'immagine ricevuta. In questo modo l'utente quando ha necessità di recuperare l'immagine, effettuerà una ricerca. Quando il sistema ha compreso il tipo di dato richiesto, provvederà a inviarlo. I requisiti funzionali sono stati rispettati, per la valutazione dei requisiti sull'usabilità ci si è basati sui feedback positivi rilasciati da varie persone alle quali si è fatto testare il sistema. I problemi non sono mancati ma si è riusciti a risolverli. Nel complesso ci si può ritenere soddisfatti del risultato ottenuto.

7.1 Sviluppi futuri

Il sistema rispetta i requisiti definiti in fase di analisi, ma non può ancora considerarsi un prodotto da rilasciare pubblicamente e immettere sul mercato. In quest'ottica vanno considerati una serie di accorgimenti che diventino di primaria importanza, quali l'analisi delle performance e il numero di utenti gestiti in contemporanea. Un'altra importante funzionalità, che ne diverrebbe un punto di forza, potrebbe essere l'uso massiccio di intelligenza artificiale per effettuare un tagging implicito molto più preciso ed efficace. Si pensi a un bot che riesca a capire la tipologia di ogni foto e ne ricavi qualsiasi tipo di informazione. Un'altra potenzialità intrinseca dei chatbot è la possibilità di instaurare una conversazione con l'utente che prescinda dall'uso di bottoni o altri costrutti che non fanno parte di un normale scambio di messaggi tra umani. Si consideri di aggiungere algoritmi e sistemi che gestiscano l'interpretazione del linguaggio naturale (NLP).

Bibliografia

[1] Gamma E., Helm R., Johnson R., Vlissides J., *Design Patterns, Elements of Reusable Object-Oriented Software*

[2] Frias-Martinez E., Cebrian M., Pascual J. M., Oliver N., *Explicit vs. Implicit Tagging for User Modeling* Data Mining and User Modeling Group, Telefonica Research

[3] Kasampalis S., *Mastering Python Design Patterns* Ed. Packet Publishing Ltd. 2015

[4] Python-telegram-bot: `https://python-telegram-bot.org/`

[5] Ricci A., *Modulo 2.1 modelli basati su macchine a stati finiti*, 2017

[6] Mapbox Geocoding: `https://www.mapbox.com/help/how-geocoding-works/`